Crowdsourcing General Computation

Crowdsourcing General Computation

Haoqi Zhang[*], **Eric Horvitz**[†], **Rob C. Miller**[‡], and **David C. Parkes**[*]
[*]Harvard SEAS, Cambridge, MA 02138, USA, {hq, parkes}@eecs.harvard.edu
[†]Microsoft Research, Redmond, WA 98052, USA, horvitz@microsoft.edu
[‡]MIT CSAIL, Cambridge, MA 02139, USA, rcm@mit.edu

ABSTRACT
We present a direction of research on principles and methods that can enable general problem solving via human computation systems. A key challenge in human computation is the effective and efficient coordination of problem solving. While simple tasks may be easy to partition across individuals, more complex tasks highlight challenges and opportunities for more sophisticated coordination and optimization, leveraging such core notions as problem decomposition, subproblem routing and solution, and the recomposition of solved subproblems into solutions. We discuss the interplay between algorithmic paradigms and human abilities, and illustrate through examples how members of a crowd can play diverse roles in an organized problem-solving process, serving not only as 'data oracles' at the endpoints of computation, but also as modules for decomposing problems, controlling the algorithmic progression, and performing *human program synthesis*.

Author Keywords
General computation, crowdsourcing, human program synthesis, task routing, task decomposition, control, design

ACM Classification Keywords
H.5 Information Interfaces and Presentation: Theory and Methods; F.0 Theory of Computation: General

General Terms
Algorithm, Design, Human Factors

INTRODUCTION
In recent years, games with a purpose like the ESP game [6] and task markets like Amazon Mechanical Turk (mturk.com) have become successful crowd-based systems that attract a crowd to perform a variety of tasks that are difficult for computers, yet solvable by humans. The ability to attract a crowd allows for massive parallel processes across many "human computers," leading to high throughput on tasks like image labeling, audio transcription, and product categorization. With the growth of online task markets, such parallel solving is becoming increasingly available to anyone with a task in mind, and the willingness to pay for its completion.

Although such systems allow a variety of tasks to be completed by humans, the tasks posed for solution are typically simple, and thus easy to divide and distribute across individuals. While the crowd can provide tremendous value for such applications, we would also like to leverage a crowd for classes of tasks that are inherently more complex and less easily partitioned into sets of tasks that can be separately solved, e.g., writing, planning, and coding, to name some examples. Systems like Wikipedia provide evidence that it is possible to coordinate a crowd for such complex tasks (at the same scale), but we are only scratching the surface on how a crowd can contribute to the solution of a complex problem.

As we move from parallelization based on simple partitioning and distribution to more sophisticated problem-solving procedures, we find it useful to view the coordination of a crowd's problem-solving as *programming*, and also, as *organization design*. Little et al. [3] developed Turkit, a toolkit that enables requesters to write programs executed by human workers on Mechanical Turk, showing how basic programming constructs can be applied to human computation. We are exploring opportunities to generalize the basic programming ideas explored to date, such as iterative refinement, to a wider range of algorithmic concepts and procedures developed in computer science for solving different kinds of problems. Attempts to implement a wider range of algorithmic concepts derived from computational problem solving quickly bring into focus opportunities for organization design, as people within a crowd may perform different roles based on their interests and expertise. As in traditional organizations, individual roles need not be limited to 'doing the work', but may include defining and communicating subgoals, evaluating the value of current solutions, and routing tasks to appropriate individuals.

Taking this perspective that coordinating a crowd can be thought of as programming and organization design, we are investigating the interplay between algorithmic paradigms and human abilities and interests for the purposes of *crowdsourcing general computation* that enables the solution of new classes of tasks via human computation. One direction of work in this realm is the development of task environments that enable members of the crowd to be organized (or self-organized) effectively to compose a solution to a problem. We consider well-known algorithmic methods such as

CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.

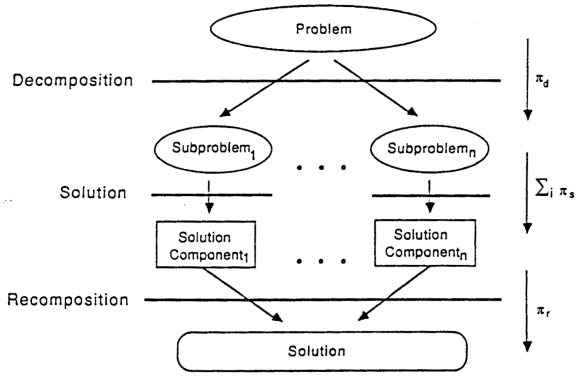

Figure 1. Diagram depicting the 'decompose, solve, and recompose' structure of divide-and-conquer algorithms, with reasoning about costs and tradeoffs associated with different phases of the problem solving process (from [1]).

divide and conquer and local search, and illustrate through examples different patterns of interaction in which members of a crowd can play diverse roles in an organized problem-solving process—where people not only serve as 'data oracles' at the endpoints of computation, but also as modules for decomposing problems, controlling the algorithmic progression, and even performing *human program synthesis*.

We identify three interrelated subareas of study, each focused on a particular way that the crowd can be harnessed in more general problem solving:

- Opportunities and challenges with applying known computational procedures to human problem solvers, recognizing the potential and need for designing algorithms whose subroutines are tailored to the crowd's abilities and interests.

- Studies of the crowd's ability to guide the control flow of an algorithm, taking advantage of the ability of people to make decisions on whether to solve the problem now or further decompose the problem, when to backtrack or to continue in a search process, and what subgoals to communicate in organizing the efforts of other contributors.

- Exploration into using a crowd as a general problem solver, where given a problem statement as input (a goal), the crowd *synthesizes* a solution to the problem by defining the problem-solving process and executing the plan.

These areas of study promise to enable new classes of applications for human computation, by providing mechanisms for involving the crowd in formulating and optimizing problem-solving plans and executing the solutions.

To highlight opportunities within the first two subareas, we shall focus on examples drawn from classic computational problems studied in algorithms and AI, which, by being well-defined and well-studied, help to illustrate methods for engaging a crowd to perform more general computational solution procedures. For the third subarea, we describe an example of how humans can help break down a goal into a plan of execution.

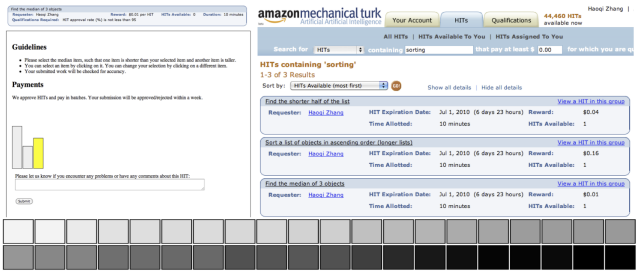

Figure 2. Snapshot of TurkSort, showing a pivot task [left], the Mechanical Turk page listing the sorting subroutines available to workers [right], and a (mostly correct) sorted list of tiles by grayscale [bottom].

ALGORITHM INSPIRED, CROWD-TAILORED

To enable effective and efficient coordination among human problem solvers, we draw on algorithmic paradigms such as divide-and-conquer for decomposing a problem into subproblems (which may be further decomposed), and for composing solutions of subproblems into solutions. Divide-and-conquer algorithms are intended for parallel processing, and are thus ideal candidates for human computation. Figure 1 provides a structural view of divide-and-conquer algorithms, wherein one can reason about the relative costs and tradeoffs associated with different phases of problem solving when deciding on a particular solution strategy.

We will focus on the familiar problem of sorting as an example. We consider how we can couple computational insights about sorting with our understanding of human abilities and interests, to design a crowd sorting task. Sorting tasks are commonplace, and include as examples, ordering favorite pictures from a family reunion, ranking restaurants in a city, and ranking the relevance of search results. Drawing on algorithmic knowledge, we may wish to use Quicksort as our sorting algorithm. Quicksort employs a divide-and-conquer procedure, where pivotal elements are selected that partition a list into elements that are ranked higher or lower than the pivot, and recursive calls are made to Quicksort on these sublists. While the subroutines are all related to ranking items, they are different from the perspective of a human computer: picking a good pivot requires finding a middle ranked item, finding a partition requires identifying everything less than a reference element, and sorting requires putting things in order. As different people may have different interests and abilities in performing theses subtasks, it may make sense to give people a *choice* as to which subroutine to perform, so as to allow for an effective division of labor. In ongoing work, we are studying a system called 'TurkSort' (see Figure 2), which crowdsources sorting tasks to workers on Mechanical Turk who contribute to the group sorting effort by finding pivots, partitioning, halving (find the smaller half), or sorting, at their choosing.

Depending on the particular sorting task, different subroutines can have different costs from the perspective of a human computer. For example, we may find that Quicksort performs poorly when sorting restaurants, because humans find it difficult to find good pivots that capture the 'median restaurant.' But subroutines of other sorting algorithms may be relatively simpler, and such differences can guide the choice of

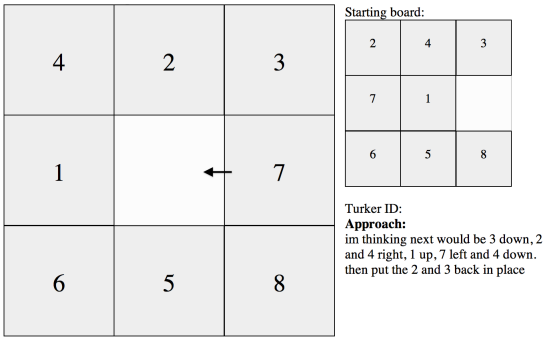

Figure 3. An iterative step in the n-puzzle game, where each Turker can make only one move. This particular Turker moved the 7 and identified a helpful sequence of moves, which was then passed down to the next worker.

the best algorithm to use. For a particular class of tasks, we might measure the crowd's performance on a number of subroutines (e.g., based on interest, accuracy, and rate of work), and use such observations to inform, or even optimize, the design of a sorting algorithm tailored to the crowd's abilities. Previous work on automatic task design [2] provide insights and methods for transforming learnings about people's performance on different tasks to designs that optimize the distribution of work in parallel workflows, and such methods can be similarly adapted for divide-and-conquer schemes.

The sorting example highlights how algorithmic insights for computer algorithms can guide the design of crowdsourcing algorithms, and the need to consider human abilities in performing various subroutines—potentially learned through experimentation. The crowdsourced algorithm's efficiency can be judged with respect to the availability and cost of human oracles of varying complexity [4], and optimized with respect to such measures. These observations point to an opportunity to provide methods and principles for custom-tailoring known algorithms for the design of particular human solution processes of interest.

HUMANS AS CONTROLLERS

In addition to harnessing the crowd's problem solving abilities within predefined modules of an algorithm, we can also leverage the crowd to guide the control flow of an algorithm. Below we describe several promising directions.

Decompose versus solve

In TurkSort, workers can choose whether to sort the current list or to decompose the list further, either within the same interface or as different tasks in the market. The base case of the recursion is defined implicitly by a decision to sort a list. More generally, providing a crowd with the options to solve a problem completely or to first decompose the problem further can be useful for a spectrum of tasks where the quality and difficulty of subtasks may be hard to determine a priori.

Transmitting solution context and subgoals

Some computational methods track and pass parameters on local and global states and on measures of progress as part of problem solving. Human computation may face similar challenges with sharing context among workers about problem-solving strategy and state, particularly when the computation is divided into small pieces performed by many workers. Unless a decomposition is defined and that context is shared, it may be hard for people to contribute effectively. For example, imagine the problem of writing code or writing an article, where we wish to enable a crowd to contribute *iteratively*, with each worker expected to make only a small contribution. A worker would have to know enough about what it is they should work on to work effectively on a subgoal at hand, and to know how the subgoal may fit within the overall aim. But if the cost of understanding the context dominates the time that the worker is willing to contribute, then this kind of collaboration becomes costly, ineffective, or impossible.

In ongoing work, we are studying such challenges with the 8-puzzle, where the goal is to slide tiles on a board until the numbers on the tiles are in order. Instead of having one person solve the puzzle, we wish to understand how workers may deal with limited problem-solving context by allowing each worker to make just one move. With limited context, workers may get stuck on difficult board positions, where thrashing can occur with states being revisited by successive contributors. We seek to understand whether it is possible to pass along a small amount of context from worker to worker—with no formal agreements on subgoals—while still making progress towards the goal. In particular, we are exploring a version of the game in which a worker is provided with the last person's move and their short explanation for making the move. We are finding that communication can be very useful in some cases. Note Figure 3, where a Turker noticed a particular path forward and pointed this out. Had he not contributed that action and highlighted the path, the problem would not have been as easily solved and many more steps would have been taken. More importantly, the ability to pass on context allows the next player to better know what to do, raising the probability that progress will be made toward the solution.

Controlling search processes

In local search problems like the 8-puzzle, the ability to guide the search process towards good neighborhoods and to backtrack when necessary are important components of an effective search method. With a human computation approach to these problems, people can assess the current solution state, decide what neighborhood to search in, and to backtrack when further improvements from the current state are unlikely.

TOWARDS HUMAN PROGRAM SYNTHESIS

As the crowd engages in algorithmic control, human computers are no longer limited to providing data output for predefined modules, but can fill in parameters of the algorithm itself and make evaluative decisions, defining the best paths through a solution space. An interesting question is whether a crowd can go beyond algorithm control, towards the notion of *synthesis*. In machine computation, *program synthesis* considers the use of appropriate design tactics to systematically derive a program based on a problem specification. For example, the synthesis of a divide-and-conquer algorithm [5]

Help someone to understand what to do to best assist a loved one who has just been diagnosed with Lou Gehrig's disease

- Get it diagnosed, through Electromyography test.
- Take Physical therapy and rehabilitation treatments.
- research the disease
- help to overcome the disease.
- Listen to your love one, don't not pity
- Read "Tuesdays with Morrie"
- join a support group
- Talk to the loved one's doctor for advice and knowledge about the disease, its progression, and possible treatments.

Figure 4. A goal is decomposed into a list of actionable steps, produced iteratively by a group of Turkers.

may involve the derivation of a tree of specifications, where leaves in the tree represent subproblems for which solutions can be readily provided, and instructions for recomposition are also derived. Taking the analogy to the crowd, we seek to enlist a crowd in both program synthesis and program execution by considering the types of problems that can be solved by the crowd, and engaging the crowd in the process of constructing an overall plan for the problem-solving process and for executing the plan. Such plans can include decomposing a problem into subproblems, solving the subproblems, and then recomposing the solved subproblems into solutions.

This problem is intriguing from a design perspective, and can also lead to novel applications. To be concrete, let us consider the problem of constructing a crowd-based advice system that takes as input a goal in natural language and provides a solution that helps to accomplish that goal. The goal itself can be anything, e.g., 'to plan a weekend in Seattle for a family with a dog from New York,' or, 'to help someone to understand what to do to best assist a loved one who has just been diagnosed with Lou Gehrig's disease.' To guide the crowd synthesis process, we consider a top-down approach in which workers are asked to define subgoals, which can then be acted upon or further decomposed. In on-going work, we initiate the first step in this process by requesting Turkers to each provide one action step towards helping someone with the goal. Turkers are shown the list of suggestions provided so far, and are asked to explain why their suggested step is helpful for the goal. Figure 4 shows the steps generated by Turkers for the goal of helping a loved one with Lou Gehrig's disease. We see a diverse set of suggestions, some of which are highly relevant (e.g., 'read Tuesday with Morrie') and specific enough to be directly acted upon by the person who had originally requested the advice. Other steps, e.g., 'research the disease', are more involved and should be decomposed into more specific actionable steps. Leveraging the notion that the crowd can control the problem-solving process by making decompose vs. solve decisions, we can ask workers for each step returned whether a step is actionable as is (by a Turker or the original requester) or requires further decomposition. As we decompose further and subproblems become solvable, (e.g., finding articles from particular trustworthy sources, drawing summaries, and so on), workers can then provide solutions for these problems. We can imagine crowdsourcing the recomposition process as well, which may need to be broken down into subproblems itself, depending on complexity.

In this example, the design of the algorithmic structure (e.g., how steps are requested iteratively, how decompose vs solve is used) frames the solution process, based on which workers provide the actual decomposition that defines the strategy for solving the particular problem. Although the approach may require refinement for particular applications, it provides an overall sketch for a potentially useful, crowd-powered system.[1] Understanding how to design effective patterns of interactions for controlling the synthesis process is an important area for future work, and should draw on understanding of peoples abilities to perform synthesis-related actions such as suggesting subgoals and collating ideas.

ON EXPERTISE

As we move toward crowdsourcing general computation, the notion of expertise becomes more prominent as roles people play are more diverse and specialized. The ability to identify expertise, and to reward individuals for providing meta-expertise (e.g., controlling the algorithmic process, routing to others who are experts), can allow us to solve problems we otherwise would not be able to solve with a crowd. As such, making use of expertise serves as a complementary agenda for the study of crowdsourcing general computation.

CONCLUSION

We presented ongoing research and a larger research agenda moving forward for harnessing human computation in general problem solving. We introduced three subareas of study that focus on particular ways that a crowd can be harnessed for general problem solving. We see great opportunities for leveraging algorithmic paradigms in the design of human-computation systems, and much research ahead on principles and methods that can enable new classes of application.

ACKNOWLEDGMENTS

We thank Edith Law for many helpful discussions, and Paul Koch for development support on the TurkSort system.

REFERENCES

1. Horvitz, E. J. Problem-solving design: Reasoning about computational value, tradeoffs, and resources. In *Proc. NASA Artificial Intelligence Forum* (1987).
2. Huang, E., Zhang, H., Parkes, D. C., Gajos, K. Z., and Chen, Y. Toward automatic task design: a progress report. In *Proc. HCOMP '10* (2010).
3. Little, G., Chilton, L. B., Goldman, M., and Miller, R. C. Turkit: tools for iterative tasks on mechanical turk. In *KDD-HCOMP '09* (2009).
4. Shahaf, D., and Amir, E. Towards a theory of ai completeness. In *Proc. Commonsense'07* (2007).
5. Smith, D. R. Top-down synthesis of divide-and-conquer algorithms. *Artificial Intelligence 27,1* (1985), 43–96.
6. von Ahn, L., and Dabbish, L. Labeling images with a computer game. In *CHI '04* (2004).

[1] A simple yet interesting human synthesis system is 'ifWeRanTheWorld' (www.ifwerantheworld.com), in which people post goals they would like to accomplish and friends and like-minded people suggest and perform microactions toward achieving the goal.